

## **REMARKS**

Claims 1-26 are pending in the present application. By this Response, claims 3, 13 and 19 are amended to correct errors as suggested by the Examiner. Independent claim 1 is amended to recite receiving a call from an application to perform an operation, in response to receiving the call, automatically identifying a routine to perform the operation; in response to receiving the call, automatically identifying a keystore containing the key; and creating a data structure used by the routine to execute the operation, wherein the data structure includes parameters of the call received from the application. Claims 10, 17, and 26 are amended to be consistent with amended independent claim 1. Reconsideration in view of the above amendments to claims and the following remarks is respectfully requested.

Amendments are made to the specification to correct errors and to clarify the specification. No new matter is added as a result of the amendments to the specification.

Also, Applicants submit proposed corrections to element 510 of formal drawing Figure 5 labeled “REPLACEMENT SHEET” as suggested by the Examiner.

### **I. Objection to Claims, Claims 3, 13 and 19**

The Office Action states that claims 3 and 19 are objected to because of informalities of “routing” should be replaced with “routine”. By this Response, dependent claims 3 and 19 are amended to recite “wherein the routine and the keystore are identified using the data structure”. (emphasis added)

The Office Action also states that claim 13 is objected to because the term “form” should be replaced with “from”. By this Response, dependent claim 13 is amended to recite “wherein the keystore is within a plurality of keystores and wherein the keystore application program interface layer identifies the keystore from the plurality of keystores” (emphasis added).

Thus, in view of the above amendments, Applicants respectfully request the withdrawal of objection to dependent claims 3, 13 and 19.

## II. 35 U.S.C. § 102(b), Alleged Anticipation, Claims 1-5, 7-9, 17-26

The Office Action rejects claims 1-5, 7-9 and 17-26 under 35 U.S.C. § 102(b) as being anticipated by Sun Microsystems's JAVA Platform v1.2 (JDK 1.2), published December 1999, partly described in documents "jarsigner – JAR Signing and Verification Tool" (jarsigner), "VM Spec Structure of the Java Virtual Machine" (VM Spec) and "Java Platform 1.2 API Specification: Class KeyStore" (API Spec)<sup>1</sup>. This rejection is respectfully traversed.

As to claims 1, 5, 17, 21 and 26, the Office Action states:

Regarding claims 1, 5, 17, 21 and 26, Sun discloses the jarsigner tool as an example implementation of JDK 1.2. The jarsigner tool/routine is called by a user who identifies the keystore containing a key to be used (see jarsigner, page 4, § KeyStore Aliases). Jarsigner performs operations such as signing Java Archive (JAR) files and verifies signatures (see jarsigner, page 1, § Description). While the jarsigner reference does not disclose creating a data structure and sending the data structure to the routine, whenever a Java method is instantiated, a Java Virtual Machine frame/data structure is created to store data and partial results used by the method/routine (see § 3.6, VM Spec).

Office Action dated November 10, 2003, page 4.

Amended independent claim 1, which is representative of claims 10, 17, and 26 with regard to similarly recited subject matter, now recites:

1. A method in data processing system for performing an operation using a key comprising:
  - receiving a call from an application to perform the operation using the key;
  - in response to receiving a call, identifying a routine to perform the operation;
  - in response to receiving a call, identifying a keystore containing the key;
  - creating a data structure used by the routine to execute the operation, wherein the data structure includes parameters of the call received from the application;
  - sending the data structure to the routine.(emphasis added)

---

<sup>1</sup> It is noted the Office Action rejects claims 1-5, 7-9, 17-26 under 35 U.S.C. § 102(b) based on three separate references that appear to be directed to the same product. However, Applicants respectfully submit that the rejection would be more proper if under 35 U.S.C. § 103(a), as opposed to 35 U.S.C. § 102(b).

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 21 U.S.P.Q.2d 1031, 1034 (Fed Cir. 1994). Applicants respectfully submit that Sun does not teach every element of the claimed invention arranged as they are in amended claims 1, 10, 17 and 26. Specifically, Sun does not teach in response to receiving a call, automatically identifying a routine to perform the operation; in response to receiving a call, automatically identifying a keystore containing the key; and creating a data structure used by the routine to execute the operation, wherein the data structure includes parameters of the call received from the application.

In the jarsigner reference, Sun teaches a tool named jarsigner, which is used for signing Java archive files (JAR) and verifying signatures and integrity of the signed JAR files. Jarsigner uses key and certification information from a keystore to generate digital signatures for JAR files. A keystore is a database of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. Jarsigner uses an entity's private key to generate a signature. The signed JAR file contains a copy of a certificate from the keystore for the public key corresponding to the private key used to sign the file. Jarsigner also verifies the digital signature of the signed JAR file using the certificate inside (Description, page 1).

In the VM Spec reference, Sun teaches a specification for the Java Virtual Machine (JVM). Specifically, in sections 2.1 and 3.6, Sun teaches how methods operate in the Java environment, which includes handling of exceptions when a Java program violates the semantic constraints of the Java language. In addition, Sun teaches the use of frames to store data and partial results when a Java method is invoked.

In the API Spec reference, Sun teaches a class named KeyStore, which is a class under the java.security package. The KeyStore class represents in-memory collection of keys and certificates and manages two types of entries: key entry and trusted certificate entry. Each entry in the KeyStore is identified by an "alias" string. The Keystore also

includes a number of methods for accessing the entries, for example, getting and setting a key, getting and setting a certificate.

However, Sun does not teach in response to receiving a call, automatically identifying a routine to perform an operation; in response to receiving a call, automatically identifying a keystore containing a key; and creating a data structure used by the routine to execute the operation, wherein the data structure includes parameters of the call received from an application, as recited in claim 1. The Office Action alleges that Sun teaches a routine called by a user who identifies the keystore containing a key to be used, on page 4 of the jarsigner reference, which reads as follows:

All keystore entities are accessed via unique aliases. When using jarsigner to sign a JAR file, you must generate the signature. For example, the following will sign the JAR file named "MyJarFile.jar", using the private key associated with the alias "duke" in the keystore named "mystore" in the "working" directory on the C drive.

Jarsigner -keystore c:\working\mystore -storepass mypass -keypass dukekeypasswd MyJarFile.jar duke

(page 4, jarsigner)

In the above section, as the Examiner pointed out, Sun teaches a routine that is called by the user which signs a JAR file. Sun does not teach in response to receiving a call, automatically identifying a routine to perform the operation. To the contrary, Sun teaches that the user has to call the routine, in the above example, the jarsigner, in order to perform the operation of signing the JAR file. The present invention solves this problem of handling calls from a user by automatically identifying the routine for the user to perform the operation in response to receiving a call. The user of the present invention does not have to identify what routine to call based on the operation. Thus, Sun does not teach such features as recited in claim 1.

In addition, Sun does not teach in response to receiving a call, automatically identifying a keystore containing the key. From the above section, Sun teaches that the user has to identify a routine to sign a JAR file. In order to sign a JAR file, the user also has to enter in the command line (when the call is made) the location of the keystore. Thus, Sun does not teach in response to receiving a call, identifying a keystore. To the contrary, in the presently claimed invention the keystore is identified automatically in

response to receiving the call without requiring the user to enter the keystore location. Thus, Sun does not teach such features as recited in claim 1.

Furthermore, Sun does not teach creating a data structure used by the routine to execute the operation, wherein the data structure includes parameters of the call received from an application. The Office Action alleges that while Sun does not disclose creating a data structure and sending the data structure to the routine, when a Java method is instantiated, a Java Virtual Machine frame/data structure is created to store data and partial results used by the method/routine. The Office Action alleges that Sun teaches these features at section 3.6 of the VM Spec, which reads as follows:

A Java Virtual Machine frame is used to store data and partial results, as well as to perform dynamic linking, to return values for methods and to dispatch exceptions. A new frame is created each time a Java method is invoked. A frame is destroyed when its method completes, whether that completion is normal or abnormal....Each frame has its own set of local variables and its own operand stack.

(Section 3.6, VM Spec)

However, in the above section, Sun merely teaches a data structure that is created at run time to store data when a method/routine is invoked on the Java Virtual Machine. The frame that Sun teaches is local to the routine. When a routine is invoked, the frame is created; when the routine is complete, the frame is destroyed. Each frame also has a set of local variables for storing data used by the routine during execution of the routine. However, the local variables of Sun are different from parameters of the present invention in that the parameters of the present invention are received from an application which is outside of the routine. The parameters are values received with the call from an application. The parameters exist outside of the routine even before the routine is invoked and continue to exist after the routine is terminated. To the contrary, the local variables in each frame live and die with the routine. Once the routine is terminated, the frame no longer exists. Therefore, Sun does not teach the same data structure as recited in claim 1.

In view of the above, Applicants respectfully submit that Sun does not teach each and every feature of independent claim 1 as is required under 35 U.S.C. § 102(b). The other independent claims 10, 17 and 26 recite similar features that are also not taught by Sun. Therefore, Applicants respectfully submit that Sun does not teach each and every

feature of claims 1, 10, 17, and 26. At least by virtue of their dependency on claims 1, 10, and 17, respectively, Sun does not teach each and every feature of dependent claims 2-5, 7-9, and 18-25. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1-5, 7-9, and 17-26 under 35 U.S.C. § 102(b).

In addition, Sun does not teach the specific features recited in dependent claims 2-5, 7-9 and 18-25. For example, with regard to claim 3, which is representative of claim 19 with regard to similarly recited subject matter, Sun does not teach that the routine and the keystore are identified using the data structure. The Office Action alleges that Sun teaches, at section 3.6 of the VM spec which is reproduced above, that each frame/data structure represents a particular method and comprises memory space for local variables and an operand stack/configuration. The stack frame is created/initialized as part of the method instantiation, after which execution occurs.

However, Sun does not teach that the routine and the keystore are identified using the data structure. Sun merely teaches, in the above section, a frame allocated to store data that is local to the routine. As described above, the frame is not the same as the data structure of the present invention. In addition, nowhere in the above section does Sun teach that the routine and the keystore are identified using the data structure, as recited in dependent claim 3. Since the frame of Sun is created only when the routine is invoked, the frame cannot be used to identify the routine without first invoking the routine. Furthermore, the frame of Sun only identifies local variables used during execution of the routine, it does not identify a keystore. The local variables of Sun live and die with the routine. The keystore of the present invention is a separate object that exists outside of a routine, such as a virtual keystore, a keystore located in hardware, or a keystore located in a removable device. Therefore, Sun does not teach wherein the routine and the keystore are identified using the data structure.

With regard to dependent claim 5, which is representative of claim 21 with regard to similarly recited subject matter, Sun does not teach a keystore that is one of a virtual keystore, a keystore, an adaptor, and a keystore in a smart card. While Sun teaches in the jarsigner reference that a keystore may be specified with an alias, nowhere in any of the references does Sun teach the keystore as one of a virtual keystore, an adapter, and a keystore in a smart card. To the contrary, on page 4 of the jarsigner reference, which is

reproduced above, Sun teaches a keystore in a working directory on the C drive of the user's computer. Sun does not teach or suggest a keystore that is a virtual keystore or a keystore in a smart card or an adaptor, both of which are located outside of the user's computer. Thus, Sun does not teach such features as recited in claim 5.

With regard to dependent claim 9, which is representative of claim 25 with regard to similarly recited subject matter, Sun does not teach responsive to receiving the result, performing any necessary updates to objects in the keystore. While Sun teaches, on pages 7 and 8 of the API Spec, methods "setKeyEntry" and "deleteEntry" to set and remove entries in a keystore, Sun does not teach performing the updates to the keystore responsive to receiving the result. In the jarsigner reference, Sun only teaches signing a JAR file and verifying the signature and integrity of the signed JAR file using a keystore. However, nowhere in either of the references does Sun teach performing necessary updates to objects in the keystore responsive to receiving a result. Sun only teaches using entries currently exist in the keystore to sign and verify a JAR file. Thus, Sun does not teach such features as recited in claim 9.

Thus, in view of the above, Applicants respectfully submit that Sun does not teach each and every specific feature recited in the dependent claims in addition to the features of their respective independent claims. Accordingly, Applicants respectfully request withdrawal of the rejection of dependent claims 2-5, 7-9, 18-25 under 35 U.S.C. § 102(b).

### **III. 35 U.S.C. § 103(a), Alleged Obviousness, Claims 6, 10-16, and 22**

The Office Action rejects claims 10-12 and 14-16 under 35 U.S.C. § 103(a) as being unpatentable over Release 1.0 of Intel's Common Data Security Architecture, partly described in "Intel's Common Data Security Architecture", by Intel Corporation (Intel), published December 1996 in view of Sun Microsystem's JAVA Platform v1.2 (JDK 1.2), partly described in "Java Platform 1.2 API Specification: Class KeyStore" (API Spec) and "VM Spec Structure of the Java Virtual Machine " (VM Spec), by Sun Microsystems (Sun). This rejection is respectfully traversed.

As to claims 10 and 11, the Office Action states:

Regarding claims 10 and 11, Intel discloses a security layer/CSSM and a plurality of cryptographic routines/CSPs accessed through a security

layer/CSSM (see page 33). Intel further discloses applications calling routines/services through a security layer/CSSM to perform cryptographic operations and receiving the results (see page 10), but lacks a keystore and a keystore application program interface layer coupled to the security layer. However, Intel discloses a “Java to CSSM Wrapper”/keystore application program interface layer (see page 33). Referring to the API Spec, Sun discloses a keystore class for creating “keystores”, enabling the storage and management of cryptographic keys (see page 1). Referring to VM Spec, Sun teaches that JDK 1.2 offers the benefits of multi-platform compatibility, small code size and user security (see § Introduction, paragraph 7 titled “The Java Virtual Machine”). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to include a keystore and “Java to CSSM Wrapper”/keystore application program interface layer in the CDSA to manage keys. One of ordinary skill in the art would have been motivated to perform such a modification, as suggested by Intel, to gain the benefits of Java’s multi-platform compatibility, small code size and user security, as taught by Sun.

Office Action dated November 10, 2003, page 6.

Amended independent claim 10 reads as follows:

10. A cryptographic system for use in a data processing system comprising:  
a security layer;  
a plurality of cryptographic routines, wherein the plurality of cryptographic routines are accessed through the security layer;  
a keystore; and  
a keystore application programming interface layer coupled to the security layer, wherein the keystore application programming interface layer receives a call from an application to perform a cryptographic operation, identifies a routine in response to receiving the call, calls the routine to perform the cryptographic operation, receives a result from the routine, and returns the result to the operation.(emphasis added)

While Applicants agree with the Examiner that Intel’s teaching lacks a keystore and a keystore application program interface layer coupled to the security layer, Applicants do not agree with the Examiner’s allegation of the “Java to CSSM wrapper” being the same as the keystore application programming interface layer as recited in claim 10. On pages 32-33 of the reference, Intel teaches CDSA’s support of the Java programming language for the Java programmers by defining a set of Java classes based on CSSM and the methods of these Java classes are implemented as native methods that invoke CSSM’s C language API. Thus, the “Java to CSSM wrapper” is a set of Java



security classes that implement native methods in C in order to support the CSSM functions. On page 34 of the reference, Intel teaches 12 Java security classes that are supported. They are: Certificate, CRL, Crypto, DataStore, Key, SecurityContext, TrustPolicy, CL Registry, CSPRegistry, DLRegistry, TRRegistry, CSSMException.

However, none of the above supported classes include a keystore or a keystore application programming interface. The keystore application programming interface of the present invention performs various functions, which include receiving a call from an application to perform cryptographic operation, identifying a routine, calling the routine to perform the operation, receiving a result from the routine and returning the result to the application. Nowhere in the Intel reference are these functions performed by any of the security classes. The Java security classes of Intel merely wrap calls from Java applications or applets into native methods written in C that may be understood by CSSM API. The Java security classes do not perform various functions, such as identifying a routine or calling the routine identified to perform cryptographic operations.

In the Java Platform 1.2 references, Sun teaches a keystore class that may be used to store and update key entries, but Sun fails to teach or suggest a keystore application programming interface that identifies a routine upon receiving a call from an application or returning results to the application. Therefore, a person of ordinary skill in the art would not be led to modify or combine these two references to reach the presently claimed invention, because neither Intel nor Java Platform 1.2 teaches a keystore application programming interface. Thus, neither Intel nor Sun teaches such features as recited in claims 10 and 11. .

As to dependent claim 15, Sun does not teach that the keystore application programming interface layer performs updates to the keystore in response to receiving the result from the routine. The Office Action alleges that Sun teaches methods such as setKeyEntry and deleteEntry for the purposes of updating the keystore on pages 7-8 of the API Spec and hence teaches the features of claim 15. As described above, neither Sun nor Intel teaches a keystore application programming interface that performs functions such as identifying a routine in response to receiving a call and calling the routine to perform a cryptographic operation. In addition, the “setKeyEntry” and “deleteEntry” in the API Spec of KeyStore are merely methods provided in the KeyStore

class for updating and deleting key entries when invoked by an entity outside of the class, for example, an application. The KeyStore class does not perform updates to the keystore on its own. As described on page 1 of the API Spec, the KeyStore class represents an in-memory collection of keys and certificates. The “deleteEntry” and “setKeyEntry” methods of the KeyStore class have to be invoked by another class, such as an application. Also in the API spec, the method “getInstance” is also provided for an entity outside of the KeyStore class, such as an application, to obtain a current instance of the KeyStore class prior to calling “deleteEntry” or “setKeyEntry” on the current instance to delete and update the entries. Thus, the KeyStore class itself may not initiate an update. To the contrary, the keystore application programming interface layer of the present invention performs updates automatically upon receiving a result from the routine. Thus, a person of ordinary skill in the art would not be led to modify or combine Sun’s teaching to reach the presently claimed invention since Sun does not teach a keystore application programming interface layer that performs updates as recited in claim 15.

As to dependent claim 13, the Office Action alleges that Intel discloses a cryptographic system with a keystore, but lacks particular multiple keystores. Knudsen teaches a keystore “contains all the information a single person (or application, or identity) needs for authentication,” (see page 79) where “single” indicates multiple user/keystores. Knudsen further teaches loading and saving keystores, indicating that multiple keystores are used (see page 81).

On page 79 of the Java Cryptography reference, Knudsen teaches a keystore key management paradigm, where a KeyStore is a handy box that holds keys and certificates. One KeyStore contains all the information in a single person (or application, or identity) needs for authentication. Knudsen teaches more than one private/public key pair that you need to manage by using a KeyManager class, which holds only a single key pair. On page 81, Knudsen further teaches a store and load method to load keystore data from an input stream and save keystore’s data to a given stream.

However, in none of the above sections does Knudsen teach that the keystore is within a plurality of keystores and wherein the keystore application programming interface layer identifies the keystore from the plurality of keystores. Knudsen only teaches that multiple key pairs maybe stored in a keystore to authenticate different

security data for a person, an application or an identity. Knudsen also teaches storing and loading data from a keystore. However, nowhere in the above sections does Knudsen teach a plurality of keystores or identifying a keystore within the plurality of keystores using a keystore application programming interface layer. Therefore, a person of ordinary skill in the art would not be led to modify or combine Knudsen's teaching with the Sun references to reach the presently claimed invention, because nowhere in either of the references teach a plurality of keystores and identifying a keystore within the plurality of keystores using a keystore application programming interface layer.

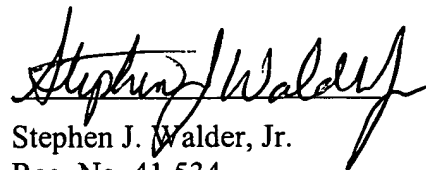
In view of the above, Applicant respectfully submits that neither Sun, Intel nor Knudsen teaches or suggests all of the features as recited in claims 6, 10-16, and 22. Accordingly, Applicant respectfully requests withdrawal of the rejection of claims 6, 10-16, and 22 under 35 U.S.C. § 103(a).

#### IV. Conclusion

It is respectfully urged that the subject application is patentable over Java Platform 1.2 by Sun Microsystems in view of Release 1.0 of Common Data Security Architecture by Intel, and further in view of Java Cryptography by Jonathan Knudsen and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

Respectfully submitted,

DATE: February 10, 2004



Stephen J. Walder, Jr.  
Reg. No. 41,534  
Carstens, Yee & Cahoon, LLP  
P.O. Box 802334  
Dallas, TX 75380  
(972) 367-2001  
Attorney for Applicants

SJW/im



09/494,876

Crane et al.

Method and Apparatus for Managing  
Keys for Cryptographic Operations

REPLACEMENT  
SHEET

3/3

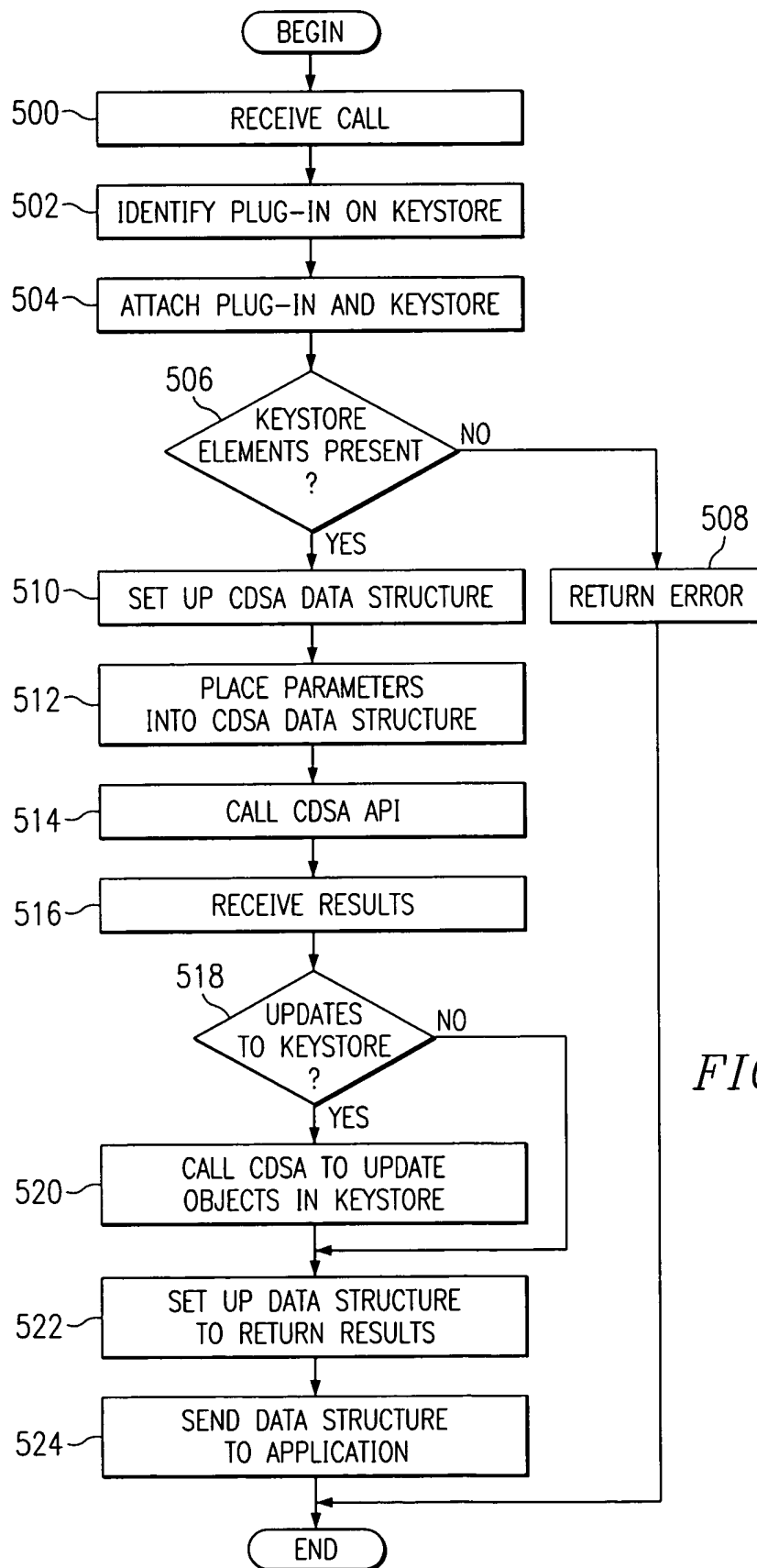


FIG. 5